

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE**

---

In the Matter of the Application of: David M. Chess et al.

Serial No.: 10/696,200

Confirmation No.: 7325

Filed: October 28, 2003

For: System Method and Program Product for Detecting Malicious Software

Examiner: Daniel L. Hoang

Group Art Unit: 2136

Attorney Docket No.: GB920030050US1

---

Commissioner for Patents

P.O. Box 1450

Alexandria, VA 22313-1450

**APPEAL BRIEF**

Sir:

In response to the Office Action of December 14, 2007, and in support of the Notice of Appeal file on March 14, 2008, Applicants respectfully submit this Appeal Brief.

**(I) Real Party in Interest**

The real party in interest for this Application is assignee INTERNATIONAL BUSINESS MACHINES CORPORATION of Armonk, NY.

**(II). Related Appeals and Interferences**

There are no related appeals or interferences.

**(III). Status of Claims**

Claims 1-14 stand rejected under 35 USC 102 as being anticipated by US Patent Publication Number 20020073323 to Jordan. Claims 1-14 are being appealed.

**(IV). Status of Amendments**

No amendments have been made to the claims.

**(V). Summary of claimed subject matter****(V.A) Claim 1**

Claims 1 is directed to a method (Figs. 2B, 6, 7A-C) for detecting malicious software within or attacking a computer system (page 1 lines 7-9, page 3 lines 13-14, page 4 lines 1-18, page 7 lines 3-10). The method comprises executing a hook routine (step 211 in Fig.2B, 600, 610, 620 Fig. 6, page 4 lines 2-10, page 7 lines 26-31, page 9 lines 19-20, page 11 lines 13-31, page 12 lines 1-30, page 13 lines 7- 30, page 14 lines 7-31, page 15, lines 1-25) in response to a system call (step 206 in Fig 2B, steps 601, 611, 621 in Fig. 6, page 4 lines 5-6, page 9 lines 17-20 and 26-28, page 11 lines 13-15 and 27-29 and 31, page 12 lines 1-3 and 5-8 and 10-12 and 14-17, page 14 lines 6-7 and 20-23, page 15 lines 1-5). The hook routine is located at the location identified by the system

call (page 4, lines 5-6, page 7 lines 27-28, page 9 lines 19-20 and 26-28, page 11 lines 13-14 and 27-28 and 31, page 12 lines 1 and 5-6 and 10-11 and 14-16, page 14 lines 6-7 and 20-21, page 15 lines 1-2). The hook routine determines a data flow or process requested by the system call (page 4 lines 6-7, page 10 lines 12-24, page 11 lines 13-15 and 27-29, page 12 lines 1-3 and 5-8 and 10-12 and 14-17, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 7-9 and 21-23, page 15 lines 3-5) and another data flow or process for data related to that of the call (decision steps 603, 613 and 623 in Fig. 6, page 4 line 7, page 10 lines 15-19, page 11 lines 30-31, page 12 lines 4-5 and 9-10 and 13-14 and 17-19 and 24-26, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 9-14 and 23-29, page 15 lines 5-10), and automatically generates a consolidated information flow diagram showing the data flow or process of the call and the other data flow or process (300 in Fig. 3, 330 in Fig. 4, 500 in Fig. 5, steps 604, 605, 614, 615, 624, 625 in Fig. 6, page 4 lines 8-9, page 10 lines 12-24, page 11 lines 1-31, page 12 lines 1-28, page 13 lines 7-27, page 14 lines 1 and 14-15 and 27-29, page 15 lines 8-10). Then, the hook routine calls a routine to perform the data flow or process requested by the system call of step 608 (step 212 of Fig. 2B, step 628 of Fig. 6, page 4 lines 9-10, page 11 lines 26-27, page 12 lines 28-30, page 13 lines 27-30, page 14 lines 16-17 and 29-30, page 15 lines 10-11).

### **(V.B) Claim 2**

Claim 2, which depends from claim 1 is directed to a user monitoring the information flow diagram and comparing the data flow process with a data flow or process expected by the user (page 15 lines 13-25).

### **(V.C) Claim 3**

Claim 3, which depends from claim 1, is directed to a process according to its parent claim and wherein the information flow diagram 300, 330, 500 illustrates locations of the data (Fig. 3 - file 301, memory 303, register 304, register 308, second file 309, pages 11, 12; Fig. 4 - file A 310, bytes 311 in memory 312, file B 313, memory 314,

register 316, bytes 317 in memory 318, file C 319, memory 320 and socket 321, page 14; Fig. 5 – memory locations 502,504, 506, pages 14, 15) at stages of a processing activity.

**(V.D) Claim 4**

Claim 4, which depends from claim 1 is directed to a process according to its parent claim and wherein the system call (step 205 in Figs. 2A, 2B, page 9 lines 11-12) is selected from the set of: open file, copy file to memory, copy memory to register, mathematical functions, write to file, and network or communication functions (page 8 lines 25-29, page 10 lines 8-10).

**(V.F) Claim 5**

Claim 5, which depends from claim 1 is directed to a process according to its parent claims and wherein the system call is a software interrupt of an operating system (step 202 in Figs. 2A, 2B, page 8 lines 9-18, page 9 lines 8-10 and 17-19, page 10 lines 8-10).

**(V.F) Claim 8**

Claim 8 is directed to an apparatus (Fig. 1B, page 7 lines 19-24) for detecting malicious software within or attacking a computer system. The apparatus comprises a computer system 110 (Fig. 1A, page 7 line 19) with a display screen 120 (Fig. 1, page 7 line 19).

The computer system executes a program of instructions (Fig. 2B, Fig. 6, Fig. 7A, Fig. 7B, page 1 lines 7-9, page 3 lines 13-14, page 4 lines 1-18, page 7 lines 3-10). The program comprises executing a hook routine (step 211 in Fig.2B, 600, 610, 620 Fig. 6, page 4 lines 2-10, page 7 lines 26-31, page 9 lines 19-20, page 11 lines 13-31, page 12 lines 1-30, page 13 lines 7- 30, page 14 lines 7-31, page 15, lines 1-25) in response to a system call (step 206 in Fig 2B, steps 601, 611, 621 in Fig. 6, page 4 lines 5-6, page 9 lines 17-20 and 26-28, page 11 lines 13-15 and 27-29 and 31, page 12 lines 1-3 and 5-8

and 10-12 and 14-17, page 14 lines 6-7 and 20-23, page 15 lines 1-5). The hook routine is located at the location identified by the system call (page 4, lines 5-6, page 7 lines 27-28, page 9 lines 19-20 and 26-28, page 11 lines 13-14 and 27-28 and 31, page 12 lines 1 and 5-6 and 10-11 and 14-16, page 14 lines 6-7 and 20-21, page 15 lines 1-2). The hook routine determines a data flow or process requested by the system call (page 4 lines 6-7, page 10 lines 12-24, page 11 lines 13-15 and 27-29, page 12 lines 1-3 and 5-8 and 10-12 and 14-17, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 7-9 and 21-23, page 15 lines 3-5) and another data flow or process for data related to that of the call (decision steps 603, 613 and 623 in Fig. 6, page 4 line 7, page 10 lines 15-19, page 11 lines 30-31, page 12 lines 4-5 and 9-10 and 13-14 and 17-19 and 24-26, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 9-14 and 23-29, page 15 lines 5-10), and automatically generates a consolidated information flow diagram showing the data flow or process of the call and the other data flow or process (300 in Fig. 3, 330 in Fig. 4, 500 in Fig. 5, steps 604, 605, 614, 615, 624, 625 in Fig. 6, page 4 lines 8-9, page 10 lines 12-24, page 11 lines 1-31, page 12 lines 1-28, page 13 lines 7-27, page 14 lines 1 and 14-15 and 27-29, page 15 lines 8-10). Then, the hook routine calls a routine to perform the data flow or process requested by the system call of step 608 (step 212 of Fig. 2B, step 628 of Fig. 6, page 4 lines 9-10, page 11 lines 26-27, page 12 lines 28-30, page 13 lines 27-30, page 14 lines 16-17 and 29-30, page 15 lines 10-11).

#### **(V.G) Claim 9**

Claim 9, which depends from claim 8, is directed to a system according to its parent claims and wherein the information flow diagram 300, 330, 500 illustrates locations of the data (Fig. 3 - file 301, memory 303, register 304, register 308, second file 309, pages 11, 12; Fig. 4 - file A 310, bytes 311 in memory 312, file B 313, memory 314, register 316, bytes 317 in memory 318, file C 319, memory 320 and socket 321, page 14; Fig. 5 - memory locations 502, 504, 506, pages 14, 15) at stages of a processing activity.

#### **(V.H) Claim 10**

Claim 10, which depends from claim 8, is directed to a system according to its parent claim and wherein the system call (step 205 in Figs. 2A, 2B, page 9 lines 11-12) is selected from the set of: open file, copy file to memory, copy memory to register, mathematical functions, write to file, and network or communication functions (page 8 lines 25-29, page 10 lines 8-10).

#### **(V.I) Claim 11**

Claim 11, which depends from claim 8, is directed to a system according to its parent claims and wherein the system call is a software interrupt of an operating system (step 202 in Figs. 2A, 2B, page 8 lines 9-18, page 9 lines 8-10 and 17-19, page 10 lines 8-10).

#### **(V.J) Claim 14**

Claim 14 is directed to a program product (page 15 lines 27-31) for detecting malicious software within or attacking a computer system (page 1 lines 7-9, page 3 lines 13-14, page 4 lines 1-18, page 7 lines 3-10). The program product executes a method that comprises executing a hook routine (step 211 in Fig.2B, 600, 610, 620 Fig. 6, page 4 lines 2-10, page 7 lines 26-31, page 9 lines 19-20, page 11 lines 13-31, page 12 lines 1-30, page 13 lines 7-30, page 14 lines 7-31, page 15, lines 1-25) in response to a system call (step 206 in Fig 2B, steps 601, 611, 621 in Fig. 6, page 4 lines 5-6, page 9 lines 17-20 and 26-28, page 11 lines 13-15 and 27-29 and 31, page 12 lines 1-3 and 5-8 and 10-12 and 14-17, page 14 lines 6-7 and 20-23, page 15 lines 1-5). The hook routine is located at the location identified by the system call (page 4, lines 5-6, page 7 lines 27-28, page 9 lines 19-20 and 26-28, page 11 lines 13-14 and 27-28 and 31, page 12 lines 1 and 5-6 and 10-11 and 14-16, page 14 lines 6-7 and 20-21, page 15 lines 1-2). The hook routine determines a data flow or process requested by the system call (page 4 lines 6-7, page 10 lines 12-24, page 11 lines 13-15 and 27-29, page 12 lines 1-3 and 5-8 and 10-12 and 14-17, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 7-9 and 21-23, page 15 lines 3-5) and another data flow or process for data related to that of the call (decision steps 603,

613 and 623 in Fig. 6, page 4 line 7, page 10 lines 15-19, page 11 lines 30-31, page 12 lines 4-5 and 9-10 and 13-14 and 17-19 and 24-26, page 13 lines 7-13 and 16-21 and 23-26, page 14 lines 9-14 and 23-29, page 15 lines 5-10), and automatically generates a consolidated information flow diagram showing the data flow or process of the call and the other data flow or process (300 in Fig. 3, 330 in Fig. 4, 500 in Fig. 5, steps 604, 605, 614, 615, 624, 625 in Fig. 6, page 4 lines 8-9, page 10 lines 12-24, page 11 lines 1-31, page 12 lines 1-28, page 13 lines 7-27, page 14 lines 1 and 14-15 and 27-29, page 15 lines 8-10). Then, the hook routine calls a routine to perform the data flow or process requested by the system call of step 608 (step 212 of Fig. 2B, step 628 of Fig. 6, page 4 lines 9-10, page 11 lines 26-27, page 12 lines 28-30, page 13 lines 27-30, page 14 lines 16-17 and 29-30, page 15 lines 10-11).

**(V.K) Corresponding Structure for Means of Claim 8**

The structure corresponding to means for executing a hook routine are the computer system 110 operating a program of instructions (Fig. 2B, Fig. 6, Fig. 7A, Fig. 7B, page 1 lines 7-9, page 3 lines 13-14, page 4 lines 1-18, page 7 lines 3-10) and equivalents.

The means for displaying the information flow diagram is the display screen 120 (Fig. 1, page 7 line 19) and equivalents.

**(VI). Grounds of Rejection to be reviewed on appeal**

Each of claims 1-14 is rejected under 35 U.S.C. 102 as being anticipated by U.S. Patent Publication 2002/0073323 to Jordan (hereafter Jordan).

The questions for appeal are whether or not each of claims 1-14 is anticipated by Jordan under 35 U.S.C. 102.

**(VII). Argument****(VII.A) Principles of Law Relating to Anticipation**

The Examiner must make a prima facie case of anticipation. “A person shall be entitled to a patent unless. . . (b) the invention was patented or described in a printed publication in this or a foreign country . . . more than one year prior to the date of the application for patent in the United States.” 35 U.S.C. 102. It is settled law that each element of a claim must be expressly or inherently described in a single prior art reference to find the claim anticipated by the reference. “A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.” Verdegaal Bros. v. Union Oil Co. of California, 814 F.3d 63, 631, 2USPQ2d 1051,1053 (Fed. Cir. 1987), cert. denied, 484 U.S. 827 (1987). Inherency, however, may not be established by probabilities or possibilities. The mere fact that a certain thing may result from a given set of circumstances is not sufficient.” In re Robertson, 169 F.3d 743, 745, 49 USPQ2d 1949, 1951 (Fed. Cir. 1999)(citations and internal quotation marks omitted). The Examiner has failed to make a prima facie case of anticipation, because the claims on appeal include various elements that are not expressly or implicitly described in the reference cited (i.e., Grimm).

**(VII.B) Rejection of Claims 1, 14 under 35 USC 102 over US Patent Publication 2002/0073323 (Jordan)**

Applicants have contended that claims 1 and 14, as originally filed are allowable because they include features that are neither disclosed nor suggested by Jordan or any other references, either individually or in combination.

***(VIII.B.1) “in response to a system call, executing a hook routine at a location of said system call”***



Claims 1 and 14 include a first element “in response to a system call, executing a hook routine at a location of said system call,” and claim 14 includes the similar element “program instructions responsive to a system call for executing a hook routine at a location of said system call.”

As described in the present application, in an exemplary embodiment a system call is an operation which transfers control of a processor, such as by stopping the current processing in order to request a service provided by an interrupt handler. The interrupt handler in turn is a program code at a memory location identified by an interrupt vector table. In this example, when a program makes a system call, the system call comprises an interrupt. By use of the interrupt vector table, the system executes the software routine located at the memory location designated (or pointed to) for the particular interrupt in the interrupt vector table. In claim 1, when a program makes a system call during execution of the program, the processor is pointed to a memory location for the system call. The hook routine, as provided in claim 1, is located at the location pointed to by the system call, and is executed in response to the system call. This is significant because the programming level at which software interrupts work is the level at which many computer viruses work (see specification page 8, line 30 to page 9, line 1). Moreover, the present application clearly indicates that it is desirable to hook the lowest level calls where possible (see specification page 9, lines 4-5).

Jordan does not disclose or suggest executing a hook routine in response to a system call. In fact, Jordan does not discuss the use of any hooking routine, much less executing a hook routine at a location of the system call in response to the system call. Rather, Jordan provides for “emulating computer executable code in a subject file, and monitoring the emulation of the computer executable code and monitoring modification of memory state by the emulated code execution, to detect an attempt by the emulated code to access one or more of the restricted computer system resources.” (paragraphs 0008, 0020). Jordan is silent on how the emulation and modification of memory state are monitored, and the Examiner may not assume a method to establish anticipation. It is not sufficient that a certain thing may result from a given set of circumstances (see In Re

Robertson, 169 F.3d 743, 745, 49 USPQ2d 1949, 1951 (Fed. Cir. 1999)(citations and internal quotes omitted).

The Examiner has failed to make a prima facie case that the element “in response to a system call, executing a hook routine at a location of said system call,” is anticipated by Grimm.

**(VII.B.2) “determine a dataflow or process requested by said call”**

Claims 1 and 14 include a second element “determine a dataflow or process requested by said call.”

In the present application, when a system call is made, the hooking routine executes to monitor and display the operation of the computer system (see specification page 10, lines 1-8). The hooking routines generate an icon or other graphical representation of the current operation to be performed by the original routine (i.e., the routine called by the system call). Thus, in claims 1 and 14, the hooking routine determines the actual function to be performed by the system call (e.g., transferring data, mathematical function, deleting data, copying data, etc.). As pointed out previously, it is important to monitor the system call functions, as this is where many malicious software programs operate.

The Examiner suggests in error that Jordan discloses this feature at paragraph 20 “process of recognizing attempts (by viruses) to (gain unauthorized access to) restricted system resources...” The cited portion of Jordan provides recognition of a specific operation (an attempt to gain unauthorized access to restricted system resources). Thus, Jordan is directed to detection of a specific operation and not determining a data flow or a process flow. The difference is significant because many viruses or malicious software operate in ways other than an unauthorized access to restricted resources. In fact they may perform a series of operations that individually seem harmless, but in combination have a malicious effect.

The Examiner has failed to make a prima facie case of anticipation of the second element, “determine a data flow or process requested by said call.”

***(VII.B.3) “determine another data flow or process for data related to that of said call”***

Claims 1 and 14 include a third element, “determine another data flow or process for data related to that of said call.” The present invention provides for associating a current system call function with another system call function by matching file names or memory locations, for example. This step is important to creating a meaningful process flow to track a data flow or process, as operations that in isolation may not appear malicious, when viewed together show a malicious pattern. Malicious software may manipulate data, for example, using a series of steps which individually do not appear malicious. By following the data or process flow the danger, however, becomes apparent.

Jordan does not disclose or suggest stringing together related system call operations to track data flow or process flows. The office action suggests that this feature is provided at paragraph [0020]. However, the cited text merely recites “monitoring both the emulation of the computer executable code and the computer system memory state”. Jordan is silent as to how the monitoring is performed or what is monitored. Thus, Jordan does not disclose or suggest determining a related data or process flow.

The Examiner has failed to make a prima facie case of anticipation of the third element, “determine another data flow or process for data related to that of said call.”

***(VII.B.4) “automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process”***

Claims 1 and 14 include a fourth element, “automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process.”

The fourth element is directed to the hooking program combining the data flow of a current system call with the data flow of related system calls and presenting this data flow information in a specific useful form, namely a consolidate information flow diagram. A diagram is a graphical representation, as shown in Figs. 3, 4, and 5 and described in the specification at pages 11-15.

Jordan does not disclose or suggest generating a consolidated information flow diagram. The Examiner suggests that Jordan discloses this feature in paragraph 27 “While the program file is being emulated, monitor component 32 monitors the code execution and any modifications of memory state (step 12), and supplies to detector component 22 information regarding the emulated code execution and any modifications of memory state by the emulated code execution. Based on the information supplied by monitor component 32, detector component 33 detects an attempt by the emulated code to access one or more of the restricted computer system resources (step 13).” Jordan, according to the cited paragraph, monitors for specific operations and does not creating a flow diagram.

The Examiner has failed to make a prima facie case of anticipation with respect to the fourth element, “automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process.”

**(VII.C) Rejection of Claim 8 under 35 USC 102 over US Patent Publication  
2002/0073323 (Jordan)**

Applicants have contended that claim 8, as originally filed is allowable because it includes features that are neither disclosed nor suggested by Jordan or any other references cited in the First Office Action, either individually or in combination.

**(VII.C.1) *elements previously discussed under claim 1***

***“means responsive to a system call, for executing a hook routine at a location of said system call”***

This element is directed to a computer operating a program of instruction substantially similar to the first element argued under claims 1 and 14. These arguments will not be repeated here.

***“determine a dataflow or process requested by said call”***

This element is directed to a computer operating a program of instruction substantially similar to the second element argued under claims 1 and 14. These arguments will not be repeated here.

***“determine another data flow or process for data related to that of said call”***

This element is directed to a computer operating a program of instruction substantially similar to the third element argued under claims 1 and 14. These arguments will not be repeated here.

***“automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process”***

This element is directed to a computer operating a program of instruction substantially similar to the fourth element argued under claims 1 and 14. These arguments will not be repeated here.

**(VII.C.2) “means for displaying said information flow diagram”**

Claim 8 includes the element “means for displaying said information flow diagram.” The structure associated with this element is the display screen 120. Thus, the information flow diagram representing the information flow determined by computer 110 is displayed on display screen 120. Jordan does not describe either an information flow diagram or a display screen. Accordingly, the Examiner has failed to make a prima facie case of anticipation.

**(VII.D) Rejection of Claim 2 under 35 USC 102 over Patent Publication  
2002/0073323 (Jordan)**

Applicants have contended that claim 2, as originally filed is allowable independently of its parent claim 1, because it includes features that are neither disclosed nor suggested by Jordan or any other references cited in the First Office Action, either individually or in combination.

**(VII.D.1) “a user monitors said information flow diagram”**

Claim 2 includes the element “a user monitors said information flow diagram.” Jordan does not disclose or suggest a user monitoring an information flow diagram. A user monitoring the information flow diagram is significant because it allows the user to detect suspicious activity in real time and take remedial measures (page 15 lines 13-25). The Examiner has argued in error that this feature is provided by Jordan at paragraph 27. Jordan provides a detector component 33 that detects specific operations. A detector component is different from a user. Moreover, Jordan does not provide a flow diagram.

**(VII.E) Rejection of Claims 3, under 35 USC 102 over US Patent Publication  
2002/0073323 (Jordan)**

Applicants have contended that claims 3 and 9, as originally filed are allowable independently of their parent claims 1 and 8, respectively, because they include a feature

that is neither disclosed nor suggested by Jordan or any other references cited in by the Examiner, either individually or in combination.

**(VII.E.1) “said information flow diagram illustrates locations of said data at stages of a processing activity”**

Claims 3 and 9 include the element “said information flow diagram illustrates locations of said data at stages of a processing activity.” The present invention provides an embodiment in which the flow of data is captured and the location of the data at stages of processing is illustrated. This allows a user to quickly see where data is being transferred to, and therefore, whether data is being manipulated in an undesirable way. Jordan does not disclose or suggest illustrating locations of data during processing, but rather detection of specific operations.

The Examiner has concluded in error that the paragraph 27 of Jordan discloses this feature. Paragraph 27 does not disclose or suggest a flow diagram or illustrating locations of data related to a system call, much less a flow diagram that illustrates locations of a data related to a system call.

**(VII.G) Rejection of Claims 5, 11 under 35 USC 102 over US Patent Publication 2002/0073323 (Jordan)**

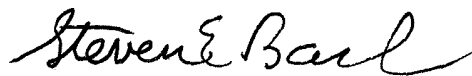
Applicants have contended that claims 5 and 11, as originally filed are allowable independently of their parent claims 1 and 8, respectively, because they include a feature that is neither disclosed nor suggested by Jordan or any other references, either individually or in combination.

**(VII.G.1) “said system call is a software interrupt of an operating system.”**

Claims 5 and 11 include the element “said system call is a software interrupt of an operating system.” Jordan does not disclose or suggest executing a hook routine in

response to a system interrupt. The Examiner suggests in error that this feature is disclosed in paragraph [0024], however this paragraph addresses how a virus works via an interrupt handler, not how a data flow or process flow diagram may be created by executing a hook routine in response to an interrupt.

Accordingly, the Examiner has failed to make a prima facie case of anticipation.

A handwritten signature in black ink, reading "Steven E. Bach". The signature is fluid and cursive, with the first letters of each word being capitalized and prominent.

Steven E. Bach  
Attorney for Applicants  
Reg. No. 46,530



**(VIII) Claims Appendix****Listing of Claims:**

1. (original) A method for detecting malicious software within or attacking a computer system, said method comprising the steps of:

In response to a system call, executing a hook routine at a location of said system call to (a) determine a data flow or process requested by said call, (b) determine another data flow or process for data related to that of said call, (c) automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process, and after steps (a-c), (d) call a routine to perform said data flow or process requested by said call.

2. (original) A method as set forth in claim 1, wherein a user monitors said information flow diagram and compares the data flow process of steps (a) and (b) with a data flow or process expected by said user.

3. (original) A method as set forth in claim 1, wherein said information flow diagram illustrates locations of said data at stages of a processing activity.

4. (original) A method as set forth in claim 1, wherein said system call is selected from the set of: open file, copy file to memory, copy memory to register, mathematical functions, write to file, and network or communication functions.

5. (original) A method as set forth in claim 1, wherein said system call is a software interrupt of an operating system.

6. (original) A method as set forth in claim 1, wherein said system call causes a processor to stop its current activity and execute said hook routine.

7. (original) A method as set forth in claim 1 wherein said system call is made by malicious software.

8. (original) A system for detecting malicious software in a computer system, said system comprising:

means, responsive to a system call, for executing a hook routine at a location of said system call to (a) determine a data flow or process requested by said call, (b) determine another data flow or process for data related to that of said call, (c) automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process, and after steps (a-c), (d) call a routine to perform said data flow or process requested by said call; and

means for displaying said information flow diagram.

9. (original) A system as set forth in claim 8, wherein said information flow diagram illustrates locations of said data at stages of a processing activity.

10. (original) A system as set forth in claim 8, wherein said system call is selected from the set of: open file, copy file to memory, copy memory to register, mathematical functions, write to file, and network or communication functions.

11. (original) A system as set forth in claim 8, wherein said system call is a software interrupt of an operating system.

12. (original) A system as set forth in claim 8, wherein said system call causes a processor to stop its current activity and execute said hook routine.

13. (original) A system as set forth in claim 8 wherein said system call is made by malicious software.

14. (original) A computer program product for detecting malicious software in a computer system, said computer program product comprising:

a computer readable medium;

program instructions, responsive to a system call, for executing a hook routine at a location of said system call to (a) determine a data flow or process requested by said call, (b) determine another data flow or process for data related to that of said call, (c) automatically generate a consolidated information flow diagram showing said data flow or process of said call and said other data flow or process, and after steps (a-c), (d) call a routine to perform said data flow or process requested by said call; and wherein

said program instructions are recorded on said medium.

**(IX). Evidence appendix**

**IX.A.**

Haas, Dr. Juergen, “system Call”, About.com, printed July 30, 2007  
([http://about.com/cs/linux101/g/system\\_call.htm?terms=system+call](http://about.com/cs/linux101/g/system_call.htm?terms=system+call))

**IX.B.**

“System Call”, Wikipedia, printed July 30, 2007  
([http://Wikipedia.org/Wiki/System\\_call](http://Wikipedia.org/Wiki/System_call))

**(X). Related proceedings appendix**

There are no related proceedings.

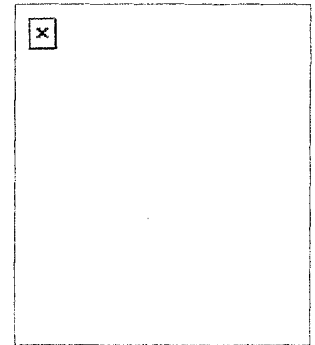
# About.com : Focus on Linux

## "system call"

From Juergen Haas,  
Your Guide to Focus on Linux.  
**FREE** Newsletter. Sign Up Now!

**Definition:** system call: The mechanism used by an application program to request service from the operating system. System calls often use a special machine code instruction which causes the processor to change mode (e.g. to "supervisor mode" or "protected mode"). From Linux Guide @FirstLinux

\* **Linux/Unix/Computing Glossary**



# System call

From Wikipedia, the free encyclopedia

In computing, a **system call** is the mechanism used by an application program to request service from the operating system.

## Contents

- 1 Background
- 2 Mechanism
- 3 The library as an intermediary
- 4 Examples and Tools
- 5 Typical implementations
- 6 External links

## Background

Application programs are a series of instructions, which manipulate data in memory. There can be many programs running on the same machine simultaneously. In addition to bare computing, the programs usually need to communicate with the real world, which consists of hardware, for observing and controlling it. Examples of accessing it are: getting the current time, allocating a memory area, reading from/writing to a file on disk, printing text on screen and even changing the processor mode. Since the machine and its devices are shared between all the programs, the access must be synchronized. Furthermore, some of the activities may fail the system or even destroy something physically. For these reasons, the access to the physical environment is strictly managed by the platform, which consists of hardware and operating system and executes the user programs. The code and data of the OS is located in a protected area and cannot be accessed/damaged by user applications. The only gate to the hardware is system calls, which are defined in OS API. These calls check the requests and deliver them to the OS drivers, which control the hardware input/output directly.

Modern processors can typically execute instructions in several differently privileged states. In systems with two levels, they are usually called user mode and supervisor mode. Different privilege levels are provided so that operating systems can restrict the operations that programs running under them can perform, for reasons of security and stability. Such operations include accessing hardware devices, enabling and disabling interrupts, changing privileged processor state, and accessing memory management units. The operating system kernel would run in supervisor mode, and user applications in user mode which has a low privilege.

With the development of separate operating modes with varying levels of privilege, a mechanism was needed for transferring control safely from lesser privileged modes to higher privileged modes. Less privileged code could not simply transfer control to more privileged code at any arbitrary point and with any arbitrary processor state; to allow it to do so could allow it to break security. For instance, the less privileged code could cause the higher privileged code to execute in the wrong order, or provide it with a bad stack.

## Mechanism

System calls often use a special CPU instruction which causes the processor to transfer control to more privileged code, as previously specified by the more privileged code. This allows the more privileged code to specify where it will be entered as well as important processor state at the time of entry.

When the system call is invoked, the program which invoked it is interrupted, and information needed to continue its execution later is saved. The processor then begins executing the higher privileged code, which, by examining processor state set by the less privileged code and/or its stack, determines what is being requested. When it is finished, it returns to the program, restoring the saved state, and the program continues executing.

Note that in many cases, the actual return to the program may not be immediate. If the system call performs any kind of lengthy I/O operation, for instance disk or network access, the program may be suspended (“blocked”) and taken off the “ready” queue until the operation is complete, at which point the operating system will again make it a candidate for execution.

## The library as an intermediary

Generally, operating systems provide a library that sits between normal programs and the rest of the operating system, usually the C library (libc), such as glibc and MS LibC. This library handles the low-level details of passing information to the kernel and switching to supervisor mode, as well as any data processing and preparation which does not need to be done in privileged mode. Ideally, this reduces the coupling between the operating system and the application, and increases portability.

On exokernel based systems, the library is especially important as an intermediary. On exokernels LibOSes shield user applications from the very low level kernel API, and provide abstractions and resource management.

## Examples and Tools

On POSIX and similar systems, popular system calls are open, read, write, close, wait, execve, fork, and kill. Many of today's operating systems have hundreds of system calls. For example, Linux has 319 different system calls. FreeBSD has about the same (almost 330).

Tools such as strace and truss report the system calls made by a running process.

In the Java platform, there is no need for the java processor to interrupt itself to make the system call safer. The effect is reached by providing a higher level of isolation by renunciation of arbitrary memory pointers, which allows the safe placement of all the code into one memory space. The Java machine is indistinguishable from its supporting library Java Runtime Environment in this platform where API consists of a set of system objects, invoking methods of which system calls are made, and no privileged instructions are needed. The approach is copied by Microsoft in the .Net platform and Singularity OS.

## Typical implementations



Implementing system calls requires a control transfer which involves some sort of architecture specific feature. A typical way to implement this is to use a software interrupt or trap. Interrupts transfer control to the kernel so software simply needs to set up some register with the system call number they want and execute the software interrupt. Linux uses this implementation on x86 where the system call number is placed in the EAX register before interrupt 0x80 is executed.

For many RISC processors this is the only feasible implementation, but CISC architectures such as x86 support additional techniques. One example is SYSCALL/SYSRET which is very similar to SYSENTER/SYSEXIT (the two mechanisms were invented by different companies, but do basically the same thing). These are "fast" control transfer instructions that are designed to quickly transfer control to the kernel for a system call without the overhead of an interrupt.

An older x86 mechanism is called a call gate and is a way for a program to literally call a kernel function directly using a safe control transfer mechanism the kernel sets up in advance. This approach has been unpopular presumably due to the lack of portability and existence of the faster instructions mentioned above.

## External links

- Linux system calls ([http://docs.cs.up.ac.za/programming/asm/derick\\_tut/syscalls.html](http://docs.cs.up.ac.za/programming/asm/derick_tut/syscalls.html)) - system calls for Linux kernel 2.2, with IA32 calling conventions
- Linux system calls (<http://www.lxhp.in-berlin.de/lhpsyscal.html>) - system calls for Linux kernel 2.6 IA32
- How System Calls Work on Linux/i86 (<http://www.tldp.org/LDP/khg/HyperNews/get/syscall/syscall86.html>)
- Sysenter Based System Call Mechanism in Linux 2.6 (<http://manugarg.blogspot.com/2006/07/sysenter-based-system-call-mechanism.html>)

*This article was originally based on material from the Free On-line Dictionary of Computing, which is licensed under the GFDL.*

Retrieved from "[http://en.wikipedia.org/wiki/System\\_call](http://en.wikipedia.org/wiki/System_call)"

Categories: FOLDOC sourced articles | Operating system technology | Application programming interfaces | Systems

- 
- This page was last modified 05:05, 24 January 2007.
  - All text is available under the terms of the GNU Free Documentation License. (See **Copyrights** for details.) Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a US-registered 501(c)(3) tax-deductible nonprofit charity.